VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvovanie individuálnej odbornej praxe

Individual Professional Practice in the Company

2015 Michal Krajňák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Krajňák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: dSoft Solutions s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Pavla Dráždilová, Ph.D.**

Konzultant bakalářské práce: Gustav Hlaváč

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestné prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 7. máj 2015 .....

Prehlásenie firmy

Súhlasím so zverejnením tejto bakalárskej práce podľa požiadaviek čl. 26, odst. 9 *Študijného a skúšobného rádu pre štúdium v bakalárskych programoch VŠB-TU Ostrava.*

V Ostrave 7. máj 2015

dSoft Solutions

IČO: 35876085
DIČ: 2021781542
IČ DPH: SK2021781542

Poďakovanie

Rád by som poďakoval vedúcej bakalárskej práce pani Mgr. Pavla Dráždilová, Ph.D. a pánovi Gustávovi Hlaváčovi zo spoločnosti dSoft Solutions, s. r. o. za cenné rady pri bakalárskej práci.

Abstrakt

Cieľom bakalárskej práce je zhrnúť pôsobenie v spoločnosti dSoft Solutions, s. r. o. V rámci odbornej praxi sa najskôr pracovalo so systémom Nagios. Úlohou bolo nainštalovať tento systém a nakonfigurovať ho na SMTP správy. Hlavnou úlohou bolo implementovanie rezervačného systému. Systém mal slúžiť zákazníkom na rezerváciu miest pri plesoch, svadbách, atď. Na jeho implementáciu sa používal klientský framework Angular.js a serverový framework Express.js.

Kľúčové slová: systém, konfigurácia, implementácia, framework, Angular.js, Express.js

Abstract

The aim of bachelor thesis is to sumarize the effect of dSoft Solutions, s. r. o. Within the technical practise was at first worked with the system Nagios. The task was to install this system and configure it for SMTP messages. The main task was to implement of reservation system. The system served the customers to reserve seats at proms, weddings, etc. For the implementation of the system was used cliental framework Angular.js and server framework Express.js.

Keywords: system, configuration, implementation, framework, Angular.js, Express.js

Zoznam použitých skratiek a symbolov

API	–	Application programming interface
atď.	–	A tak ďalej
CSS	–	Cascading style sheets
EJS	–	Embedded javaScript
HTML	–	HyperText markup language
HTTP	–	Hypertext transfer protocol
HTTPS	–	HyperText transfer protocol secure
HW	–	Hardware
ICMP	–	Internet control message protocol
IO	–	Input output
IT	–	Information technology
JSON	–	JavaScript object notation
napr.	–	Napríklad
NNTP	–	Network news transfer protocol
NPM	–	Node package manager
obr.	–	Obrázok
OS	–	Operating system
POP3	–	Post office protocol
SMTP	–	Simple mail transfer protocol
SNMP	–	Simple network management protocol
s. r. o.	–	Spoločnosť s ručením obmedzeným
TCP	–	Transmission control protocol
t. j.	–	To jest
tzn.	–	To znamená
UI	–	User interface
URL	–	Uniform resource locator

Obsah

1 Úvod

V tejto bakalárskej práci je popísané moje pôsobenie v spoločnosti dSoft Solutions, s. r. o. V spoločnosti som pracoval na pozícií programátor webových aplikácií a konkrétne som sa zaoberal programovaním na strane klienta, ako aj na strane servera. Vykonával som tiež práce v oblasti monitorovania bežiacich systémov. Každá spoločnosť väčšinou žiada od uchádzača nejakú skúsenosť. Znalosti, ktoré som nadobudnul v škole som si chcel rozšíriť o praktické skúsenosti vo firme. Monitorovanie systémov bolo pre mňa úplne novou vecou, preto som rád, že som mohol získať v tejto oblasti skúsenosť.

Na začiatku práce popisujem informácie o samotnej spoločnosti. Potom som popísal dôležité technológie, ktoré boli využívané pri vykonávaní úloh. Nie každý pozná trend dnešných technológií, ktoré sa používajú vo webe alebo pri monitorovaní systémov, preto popisom týchto technológií si ujasníme o čo vlastne ide. Po popísaní konkrétnych technológií v práci som následne spracoval zadania úloh, ktoré mi boli pridelené od firmy a samozrejme ich riešenie. Dostával som veľa podúloh, ktoré boli potrebné pri realizácií projektu z firmy. Jednotlivé úlohy som konzultoval so svojím nadriadeným, pánom Gustávom Hlaváčom.

V závere práce zhrňujem, aké predmety zo školy mi pomohli pri vykonávaní praxe. Na jej konci je zhrnuté celkové hodnotenie odbornej praxe.

2 Informácie o praxi

V tejto kapitole budú popísané informácie ohľadom firmy a taktiež tu bude zobrazený celkový priebeh odbornej praxe vo firme.

2.1 O firme

Spoločnosť dSoft Solutions, s. r. o. [3] sa zaoberá vývojom IT riešení pre širokú škálu segmentov administratívy a používa vlastnú infraštruktúru klientov (firiem), pre ktoré sa tieto riešenia vyvíjajú. Z tohoto dôvodu spoločnosť rieši monitorovanie a údržbu vlastného HW, ako i virtuálneho HW, ktorý je umiestnený ako v datacentrách, tak i na pobočkách firiem. Zároveň sa spektrum používaných OS týka Linuxových, Unixových ale podobne i Microsoft systémov. Väčšina (tzn. 98 %) systémov beží vo virtuálnych prostrediach, preto je následná zmena konfigurácie alebo premiestňovanie týchto riešení veľmi jednoduché.

Čo sa týka vývoja, tak spoločnosť dSoft Solutions, s. r. o. používa serverové jazyky Java, C#, C++, JavaScript a jazyk bash pre automatizáciu v Unix OS. Na strane klientov používa jazyky C#, JavaScript, HTML, CSS, Java GWT. Cieľom spoločnosti je však ponúknuť klientom riešenia v najnovších stabilných technologických možnostiach.

2.2 Harmonogram praxe

- 02. 10. 2014 - 03. 10. 2014
  Prebiehalo oboznámenie sa so spoločnosťou. Zaradenie do pracovného prostredia.
- 09. 10. 2014 - 10. 10. 2014, 16. 10. 2014 - 17. 10. 2014
  Práca so systémom Nagios. Systém Nagios mal byť nainštalovaný cez vzdialený virtuálny stroj, ktorý bol spustený v Bratislave.
- 23. 10. 2014 - 24. 10. 2014, 06. 11. 2014 - 07. 11. 2014
  Na nainštalovaný systém Nagios sa mala nakonfigurovať SMTP notifikácia správ.
- 13. 11. 2014 - 14. 11. 2014, 20. 11. 2014 - 21. 11. 2014
  Vývoj rezervačného systému. Začalo sa s dátovou analýzou. Systém bude postavený na databáze MongoDB.
- 27. 11. 2014 - 28. 11. 2014, 04. 12. 2014 - 05. 12. 2014
  Naprogramovanie registrácie pre klientov a možnosť prihlasovania sa. Prihlasovanie realizované prostredníctvom modulu passport-local.
- 11. 12. 2014 - 12. 12. 2014
  Naprogramovanie posielanie HTML e-mailov prostredníctvom šablóny EJS.
- 12. 02. 2015 - 13. 02. 2015, 19. 02. 2015 - 20. 02. 2015
  Naprogramovanie správy účtu pre administrátora, manažéra a klienta.

- 26. 02. 2015 - 27. 02. 2015
  Dynamické generovanie stolov, stoličiek, miestnosti, častí miestnosti na základnej stránke pre užívateľa.
- 05. 03. 2015 - 15. 04. 2015
  Prideľovanie práv a posielanie pozvánok na pridelenie práv. Vytvorenie dodatočných funkcií pre bezpečnosť systému.

3 Použité technológie

V priebehu praxe sa pracovalo s množstvom technológií, základom ale bol skriptovací jazyk JavaScript. Prostredníctvom neho sa pracovalo so serverovým frameworkom Node.js a klientským frameworkom Angular.js. Používalo sa vývojové prostredie WebStorm, ktoré bolo perfektne vybavené pre prácu na strane klienta, ako aj na strane servera. Pri práci s Node.js nám posluhovali moduly z baličkového systému NPM. Pri Node.js aplikáciach sa pracovalo s nástrojmi ako nodemon a http-server. Tieto nástroje nájdeme v NPM balíčku a slúžia na spustenie servera pre beh Node.js aplikácie. Pre potrebu ukládania dát sa aplikovala databáza MongoDB. Pri práci s Node.js nám poslúžil Node.js webový framework Express.js.

Na monitorovanie systémov sa využíval systém Nagios. Systém Nagios sa inštaloval a následne konfiguroval. Vo firme sa pracovalo prioritne pod systémom Linux. Na zdieľanie kódu sa používal distribuovaný systém riadenia revízií GIT.

3.1 JavaScript

JavaScript [2] je objektovo orientovaný programovací jazyk, využívaný pri tvorbe webových stránok. Na rozdiel od serverových programovacích jazykoch (napríklad PHP) slúžiacich na generovanie kódu samotnej stránky, JavaScript beží na strane klienta, teda v prehliadavači až po stiahnutí do Vášho počítača. JavaScript sa používa predovšetkým pre vytváranie interaktivných webových stránok. Príkladom použitia môžu byť najrôznejšie kontroly správneho vyplnenia formulárov, obrázky meniace sa po prejdení myšou, rozbaľovacie menu atď. JavaScript vyvinula spoločnosť Netscape v roku 1995 a v roku 1998 bol štandardizovaný organizáciou ISO.

JavaScript sa potom stal základom pre ďalšie programovacie jazyky, napr. ActionScript, používaný v technológiach Flash a Flash Lite.

3.2 Node.js

Node.js [8] je výkonné a veľmi efektívne programovanie v JavaScripte na strane servera. Skladá sa z V8 JavaScript engine od spoločnosti Google. Node.js vytvoril v roku 2009 Ryan Dahl. Node.js je takisto chápaný ako server, ktorý dokážeme spustiť a on bude načúvať prichádzajúcim dotazom, a zároveň je runtime prostredie, ktoré poskytuje API umožňujúce realizovať rôzne úlohy serverovského typu. Je postavený na Single Event Loop a Non Blocking IO. Single Event Loop znamená, že Node.js v zásade beží len v jednom vlákne (je možné ho spustiť aj vo viacerých vláknach, ale to je skôr špeciálny prípad). Non Blocking IO znamená, že Node.js všetky dlhotrvajúce operácie (čo sú v zásade operácie s databázou, súborovým systémom alebo sieťové operácie), vykonáva asynchrónne. To znamená, že to jedno vlákno nečaká za pomalými operáciami, ale vykoná svoj kód, spustí pomalú operáciu a začne obsluhovať ďalšiu požiadavku. Keď asynchrónna operácia skončí, NodeJS opäť vykoná kód, ktorý je reakciou na jej ukončenie a opäť sa venuje iným úlohám. Je to teda jedno vlákno, ktoré sa rýchlo a na krátku dobu prepína medzi rôznymi úlohami.

Takýmto typickým príkladom Node.js aplikácie je vytvorenie HTTP, HTTPS alebo TCP servera. Ako príklad je uvedený HTTP požiadavok GET na stránku spoločnosti.

```
var http = require('http');

http.get("http://www.dsoft.eu/", function(res) {
    console.log("Odpoved': "+ res.statusCode);

}).on('error', function(e) {
    console.log("Chybova správa: " + e.message);
});
```

3.3 Express.js

Express.js [4] je serverový webový framework pre Node.js. Poskytuje efektívne funkcie pre webové a mobilné aplikácie. S veľkým množstvom HTTP pomocných metód a softvérových komponentov, ktoré máme k dispozícií je vytváranie robustných API rýchle a jednoduché. Nachádza sa v baličkovom systéme NPM, odkiaľ sa dá stiahnúť a použiť na projekt.

Následujúci príklad ukáže použitie frameworku Express.js, ktorý bude mať na HTML stránke formulár a odošle sa metódou POST. Odoslanie sa realizuje na klientskej strane cez HTTP metódu POST a na serverovej strane sa spracuje frameworkom Express.js.

- Klientská strana

```
<form action='/form' method='post'>
   Meno <input type="text" name="Meno"><br>
        <input type="submit" value='Odoslat'>
</form>
```

- Serverová strana

```
var express = require('express');
var router = express();

router.post('/form', function(req, res) {
    console.log(req.body['Meno']);
});
```

3.4 Angular.js

Angular.js [1] je moderný Javascriptový framework určený na tvorbu dynamických webových aplikácií. Angular.js je vyvíjaný a udržiavaný niekoľkými zamestnancami spoločnosti Google. Angular.js je tiež MVC framework, čo znamená, že kód by mal byť rozdelený do menších celkov podľa oblastí, ktoré v aplikácií existujú:

- Model - udržuje štruktúru dát.
- View - stará sa o zobrazovanie údajov.
- Controller - riadi manipulovanie údajov, ako aj ich transformáciu pred tým, než sú zobrazené cez View.

3.5 Nagios

Nagios [7] je populárny open source systém pre automatizované sledovanie stavu počítačových sietí a im poskytovaných služieb. Je vyvíjaný primárne pre Linux, ale je možné ho prevádzkovať i na iných unixových systémoch. Je vydávaný pod GPL licenciou. V prípade chyby, ktorá sa vyskytne u zákazníka, je nagios schopný informovať administrátora skôr, ako to zistí daný zákazník. Nagios dokáže fungovať pre LINUX ale aj pre Windows systémy.

Nagios dokáže napríklad:

- kontrolovať stav pevného disku,
- sledovať záťaž procesora,
- monitorovať služby SMTP, POP3, HTTP, NNTP, ICMP, SNMP,
- posielať nofitikácie prostredníctvom e-mailu, SMS, pageru alebo VOIP.

3.6 GIT

GIT [5] je distribuovaný systém riadenia revízií. Zameriava sa na rýchlosť, efektivitu a použitelnosť u veľkých projektov v reálnej prevádzke. Je poskytovaný ako bezplatný software pod licenciou GNU GPL. Každý adresár Gitu je plnohodnotným repozitárom s kompletnou históriou a schopnosťami pre sledovanie revízií - nezávisle na serveru či na sieti.

Medzi hlavné výhody GIT patrí:

- distribuovaný vývoj,
- podpora nelineárneho vývoja,
- efektívne nakladanie s veľkými projektami,
- kryptografická autentizácia histórie,
- dizajn nástrojov.

GIT každému vývojárovi ponúka lokálnu kópiu celej histórie vývoja. Zmeny sa vždy kopírujú z jedného repozitára do druhého. Tieto zmeny sú importované ako dodatočné vývojové vetvy a môžu byť zlúčené rovnakým spôsobom, ako lokálna vetva. K repozitárom sa dá pristupovať cez protokol Gitu, alebo pomocou HTTP.

3.7 MongoDB

MongoDB [6] je open source dokumentová databáza, ktorá poskytuje vysoký výkon, vysokú dostupnosť a automatické škálovanie. Záznam v MongoDB je dokument, ktorý je ako dátová štruktúra zložená z páru poľa a hodnoty. MongoDB dokumenty sú podobné JSON objektom. Hodnoty polí môžu zahŕňať aj iné dokumenty, polia alebo polia dokumentov. Výhody použitia dokumentov sú:

- dokumenty (t.j. objekty) zodpovedajú natívným dátovým typom v mnohých programovacích jazykoch,
- dynamické schéma podporuje plynulý polymorfizmus,
- vložené dokumenty a polia znížujú potrebu nákladných spojení.

4 Zadané úlohy a riešenia

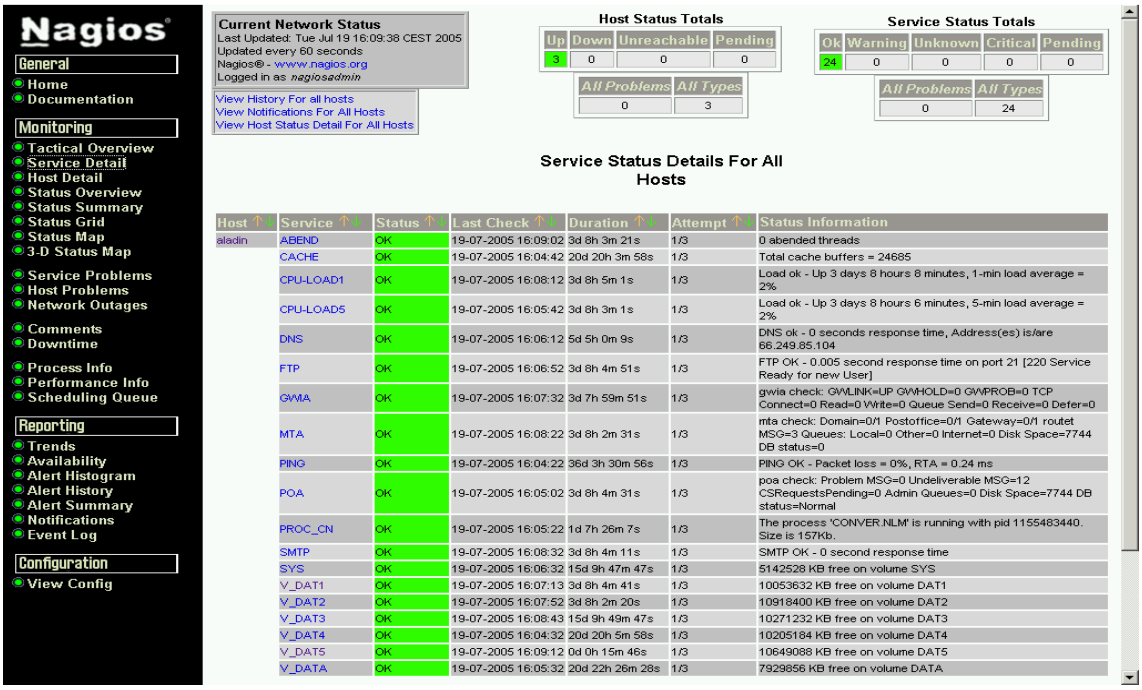
Táto kapitola obsahuje popis jednotlivých úloh, ktoré mi boli pridelené v rámci odbornej praxi. Prvá úloha spočívala v inštalácií systému Nagios a v konfigurácií na SMTP správy. Táto úloha bola menšou časťou v rámci vykonávanej odbornej praxi. Druhou úlohou a zároveň hlavným projektom bolo naprogramovanie rezervačného systému.

4.1 Systém Nagios

V rámci prvej úlohy bolo potrebné nainštalovať systém Nagios na vzdialený virtuálny stroj, ktorý fungoval v meste Bratislava. Po inštalácií systému nasledovala konfigurácia na SMTP správy.

4.1.1 Inštalácia systému Nagios

Na inštaláciu systému nagios sú potrebné určité prerekvizity, ktoré treba najskôr stiahnúť a nainštalovať. Pri inštalácií sa môže vychádzať z oficiálnej stránky systému Nagios http://www.nagios.org/. Po úspešnej inštalácií prejdeme do webového prehliadavača a ako URL zadáme http://localhost/nagios. Zobrazí sa nám tabuľka s prihlasovaním. Prihlasíme sa prostredníctvom prihlasovacích údajov, ktoré sme pri inštalácií nastavili. Po úspešnom prihlásení sa užívateľ nachádza v grafickom rozhraní systému Nagios. Na obr. 1 môžeme vidieť grafické rozhranie systému Nagios.



Obrázok 1: Grafické rozhranie systému Nagios

4.1.2 Konfigurácia SMTP servera systému Nagios

SMTP konfiguráciou bude systém Nagios schopný v prípade chyby poslať e-mail administrátorovi. Využije sa pri tom SMTP klient SendEmail na posielanie e-mailov. Po inštalácií SendEmail klienta sa prejde do konfiguráčneho súboru systému Nagios contact.cfg. V tomto súbore treba vytvoriť užívateľov, ktorým budú posielané e-maily v prípade chyby. V nasledujúcom príklade sa vytvoril užívateľ nagiosadmin, ktorému budú na zadaný e-mail chodiť správy v prípade chyby. Časový interval správ si môžeme nastaviť ľubovoľne.

```
define contact
    {
        contact_name                    nagiosadmin
        use                             generic-contact
        alias                           Nagios Admin
        email                           dsoft.tesla@gmail.com
        service_notifications_enabled   1
        host_notifications_enabled      1
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    c,w,r
        host_notification_options       d,u,r
    }
```

Kontakty, ktoré sa vytvorili je treba uložiť do spoločnej kontaktnej skupiny. Nasledujúci príklad ilustruje, ako vyzerá vytvorenie takej skupiny.

```
define contactgroup
   {
     contactgroup_name    support
     alias                Company Support
     members              helpdesk, technician1_mail
   }
```

Prejde sa do súboru commands.cfg, kde sa nadefinujú notifikácie. Kedže kód v súbore commands.cfg je dosť veľký a neprehľadný, ukáže sa len to najpodstatnejšie.

```
sendEmail -s smtp.gmail.com -t dsoft.tesla@gmail.com
-f dsoft.tesla@gmail.com -v -l /usr/local/nagios/var/nagios.log -u
```

Vysvetlenie použitých skratiek:

- s - jedná sa o SMTP server,
- t - e-mail komu bude zaslaná správa,
- f - od akého odosielateľa bude zaslaná správa.

Na to, aby sa vykonané zmeny prejavili v systéme Nagios, je potrebné ho reštartovať.

4.2 Rezervačný systém

Druhou úlohou a zároveň hlavným projektom bol vývoj rezervačného systému. Rezervačný systém má slúžiť pre Kultúrny dom v Čadci na podujatia, ako sú plesy, svadby a rôzne iné oslavy. So systémom budú pracovať klienti, administrátori a manažéri. Klienti sa do systému dostanú cez registráciu a následné prihlasovanie. Administrátori sa do systému dostanú cez pozvánku od hlavného administrátora, ktorý bude vytvorený cez MongoDB shell konzolu. Manažéri sa do systému dostanú taktiež cez pozvánku, ale túto pozvánku im už nemusí udeliť len hlavný administrátor, ale aj administrátori vytvorený hlavným administrátorom. Systém bude fungovať tak, že na základnej stránke budú zobrazené stoly a sedadlá, ktoré klient uvidí buď ako voľné, alebo obsadené.

Rezervovať si miesto môže až po vykonanej registrácií. Po prihlásení sa do systému mu bude umožnené rezervovať miesto. Po rezervovaní miesta sa daný stav sedadla zmení na rezervovaný a až po uhradení faktúry mu bude stav sedadla zmenený na zaplatený.

Rezervačný systém bude postavený na databáze MongoDB, to znamená, že na rozdiel od klasických relačných databáz nebudeme vidieť žiadne tabuľky a záznamy uložené v tabuľkách, ale budeme používať dokumenty (t. j. objekty), ktoré môžu mať niekoľko atribútov s danými hodnotami. MongoDB používa namiesto výrazu tabuľky, výraz kolekcie. Systém bude obsahovať následujúce kolekcie:

- Tables, Seats, States, Clients, Rooms, Parts, Profiles, Users, Confirmations.

4.2.1 Registrácia

Spracovanie registrácie sa bude riešiť najprv na strane klienta v Angular.js. Vytvoríme si controller v Angular.js, ktorý bude riešiť problematiku registrovania. Máme vytvorený registračný formulár na HTML stránke, ktorý bude obsahovať následujúce údaje:

- Meno.
- Priezvisko.
- E-mail.
- Heslo.
- Dátum narodenia.
- Pohlavie.

Po vyplnení a potvrdení registračného formuláru sa vykoná direktíva ng-submit, to znamená, že sa zavolá funkcia, ktorú sme si nadefinovali v controlleri. Vo funkcií sa spracuje službou $resource. $resource vytvorí objekt prostriedku a umožní nám komunikovať s REST zdrojmi na strane servera. Vytvoreným $resource objektom môžeme vykonať určité metódy. Metódou save sa vykoná HTTP metóda POST. Pri úspešnom spracovaní na strane servera môžeme druhým parametrom určiť, čo sa má stáť. Pri neúspešnom vykonaní môžeme tretím parametrom zachytiť chybu. Pomocou HTTP kódu chyby môžeme definovať detailnú správu pre užívateľa.

```
$scope.register = function (user) {
   var Register = $resource('/auth/registration');
   Register.save(user, function (data)
       {
         if (data.sameEmail)
           {
             $scope.messageDialog("Rovnaký e-mail");
           }
         else if (!data.comparePasswords)
           {
             $scope.messageDialog("Nezhodné heslá");
           }
         else
           {
             $scope.messageDialog("Registrácia úspešna",'/login');
           }
        },function(err)
           {
             if (err)
               {
                 $scope.messageDialog("Chyba");
               }
           });

        };
```

Na strane servera, teda v Express.js aplikácií spracúvavame dáta, ktoré nám boli poslané od klienta.

```
router.post('/registration',function(req, res)
{
  mongoose.model('clients').find({email: req.body['email'],
  roles: 'client'},{},function(err, users)
   {
     /*--Ak uz je e-mail v DB--*/
     if (users.length)
      {
        res.send({sameEmail: true});
      }
     /*--Ak e-mail nie je v DB--*/
     else
      {
        /*--Skontrolujem zhodu hesiel a ukladam do DB--*/
        if (req.body['password'] === req.body['password2'])
         {
           //Nasleduje testovanie hlavneho admina
           //hesovanie hesla
         }
        else
         {
           res.send({sameEmail: false,comparePasswords: false});
         }
       }
    }
```

Na ukladanie nových klientov využívame kolekciu Clients. Musíme teda najprv prejsť celú kolekciu a zistiť, či náhodou nejaký klient s daným e-mailom už neexistuje. Ak sme našli rovnaký e-mail, tak na server pošleme odpoveď, že sme našli rovnakú e-mailovú adresu. Ak nie je rovnaký e-mail nájdený, tak skontrolujeme zhodu hesiel. Ak je zhoda nesprávna tak znova pošleme na server odpoveď o nesprávnej zhode hesiel. Pri správnom e-mailu a hesla sa heslo zahešuje cez modul bCrypt a všetky hodnoty z registrácie uložíme do databázy do kolekcie Clients. Pri každom registrovanom klientovi pridáme do kolekcie Clients ešte následujúce atribúty:

- verifiedEmail: false,
- roles: ['client'].

Na základe atribútu roles vieme, že klient má rolu client a má len právo sa dostať do klientskej zóny a na základe atribútu verifiedEmail vieme, či klient potvrdil verifikačný e-mail alebo nie. Po úspešnom vložení sa pošle HTML verifikačný e-mail klientovi s potvrdzovacím odkazom a textom. Dáta, ktoré chceme mať v e-maily, posielame funkciou

render na EJS súbor, kde je HTML e-mail spracovaný a následne dostávame z funkcie upravený HTML e-mail o dáta.

```
var link = "http://" + req.get('host') + "/auth/verify?id=" + id;

var htmlData = {
   link: link,
   titleMale: params.email.html.titleMale,
   titleFemale: params.email.html.titleFemale,
   sex: req.body['sex'],
   role: getRole(req.body['role']),
   htmlAddress: req.body['firstName']+" "+req.body['lastName'],
   message: params.email.html.message,
   subject: params.email.html.subject,
   button: params.email.html.button,
   footer: params.email.footer
               };
res.render('registrationApproving',htmlData,function(err,html)
    {
      //Funkcia na posielanie e-mailov
    }
```

Tento upravený HTML e-mail pošleme cez modul NodeMailer klientovi. NodeMailer slúži na posielanie SMTP e-mailov. Vytvoríme objekt v ktorom nadefinujeme SMTP odosielateľa so službou g-mail.

```
var smtpTran = nodemailer.createTransport("SMTP",{
    service: "Gmail",
    auth: {
        user: "dsoft.tesla@gmail.com",
        pass: "********"
    }
});
```

Ak všetko prebehlo úspešne tak vrátime na server odpoveď s HTTP kódom 200.

4.2.2 Registrovanie hlavného administrátora

Na pridanie hlavného administrátora do systému nám poslúži MongoDB Shell konzola. Prejdeme do MongoDB Shell konzoly. Následne sa presunieme do našej databázy a pomocou príkazu createUser vytvoríme hlavného administrátora.

```
$ mongo
$ use db
$ db.createUser({user: 'kra0307@vsb.cz',pwd: '123456',
  roles:[{role: 'dbAdmin',db: 'db'}]})
```

Po vytvorení hlavného administrátora sa môžeme cez prihlasovací formulár prihlásiť a budeme presmerovaní na administrátorske rozhranie. Tento hlavný administrátor môže potom posielať pozvánky užívateľom na pridelenie roly.

Roly, ktoré môže hlavný administrátor prideľovať:

- administrátor,
- manažér,
- klient.

4.2.3 Verifikácia e-mailu

Po registrácií bude klientovi odoslaný verifikačný e-mail. Po potvrdení odkazu, ktorý sa nachádza vo verifikačnom e-maily, sa spracuje funkcia na kontrolu verifikácie. Táto funkcia je spracovaná na serverovej strane v Express.js aplikácií. Verifikačný odkaz obsahuje jedinečný parameter id, ktorý v kolekcii Clients budeme hľadať. Ak ho nájdeme, tak je jasné, že ide o správneho klienta. Atribút verifiedEmail zmeníme na true a vrátime úspešné vykonanie funkcie. Užívateľovi bude zobrazená hláška s úspešnou verifikáciou.

```
mongoose.model('clients').findById(req.query.id,{},
function(err, user)
   {

   });
```

Ak by sa jednalo o nesprávnu URL s nesprávnym id, tak vrátime neúspešné vykonanie funkcie s HTTP stavovým kódom 404.

```
res.send(404);
```

4.2.4 Prihlasovanie

Klientská časť prihlasovania sa programuje vo frameworku Angular.js a serverová časť vo frameworku Express.js. Na prihlasovanie sme si vytvorili samostatný controller, ktorý túto záležítosť rieši.

4.2.5 Klientská časť prihlasovania

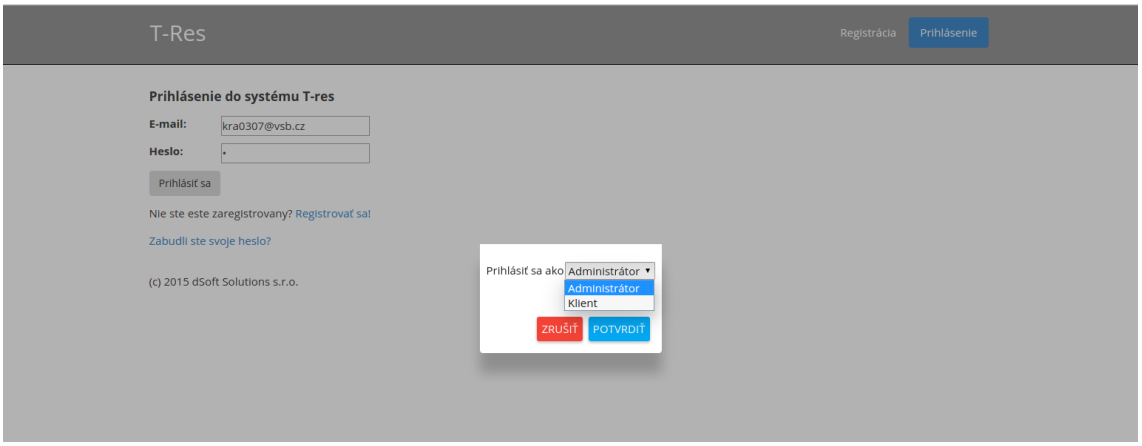
Prihlasovanie riešime na strane klienta podobným spôsobom ako registráciu. Vo formulári máme direktívu ng-submit, ktorá sa vykoná po kliknutí klientom na tlačítko prihlásiť sa. Dáta z formulára posielame na server cez prostriedok $resource s metódou save, čo znamená, že sa vykoná HTTP metóda POST.

```
$scope.Login = function(user) {
var Login = $resource('/auth/');
            Login.save(data,function(user)
            {

            },function(error)
            {

            });
}
```

Pri každom prihlasovaní musíme ešte skontrolovať, koľko má daný človek rol. Ak má len jednu rolu, tak ho automaticky prihlási na účet danej roly. Pokiaľ má viac ako jednu rolu, tak sa mu zobrazí selectBox s výberom roly. Môže si vybrať na, ktorú rolu sa chce prihlásiť. Na obr. 2 môžeme vidieť ako vyzerá taký výber účtu.



Obrázok 2: Ukážka výberu roly

Ak by sa chcel klient dostať do svojej správy účtu a nebol by ešte prihlásený, tak mu systém neumožní sa tam dostať, ale presmeruje ho na prihlasovací formulár. Táto funkcionalita je riešená cez tzv. resolve v Angular.js.

Resolve je vlastnosť, ktorá môže obsahovať route. Táto vlastnosť musí skončiť úspešné aby sa mohol vykonať daný route.

```
var deferred = $q.defer();
/*--Testujeme, ci je uzivatel prihlaseny--*/
$http.get('/auth/loggedin').success(function (user)
   {
     /*-Prihlaseny a s pravami-*/
     if (user && localStorage.getItem('role') === 'client')
      {
        if (user.role.indexOf(localStorage.getItem('role'))>-1)
         {
           deferred.resolve(user);
         }
        else
         {
           deferred.reject();
           $window.location.assign('/home');
         }
      }
     /*--Neprihlaseny--*/
     else
      {
        deferred.reject();
        $window.location.assign('/login?from=' + $location.url());
      }
    });
```

V resolve sa vytvorí objekt typu $q.defer, čo je vlastne služba, ktorá nám pomáha spúšťať funkcie asynchrónne. Táto služba má metódu resolve(), ktorá, keď sa zavolá, tak nám umožní prejsť na danú URL. V prípade metódy reject() je nám zamietnuté sa dostať na požadovanú URL. Na to, aby sme rozhodli, kedy prebehne resolve, a kedy reject použijeme službu $http a jej metódu get. Touto metódou sa nám na strane servera spracuje metóda get s URL, ktorú sme uviedli ako parameter. V danej metóde otestujeme, či je klient prihlásený, a ak áno, tak klientovi je umožnené sa dostať na danú URL.

```
res.send(req.isAuthenticated() ? req.user : false);
```

4.2.6 Serverová časť prihlasovania

Na serveru pre prihlasovanie používame modul passport-local.

```
router.post('/',passport.authenticate("local"),function(req,res)
{

});
```

Passport-local využíva funkciu use, ktorá preberá e-mail a heslo. Tieto dva parametre využívame pri prechádzaní kolekciou Clients. Musíme skontrolovať či nejde náhodou o administrátora, manažéra alebo klienta. Ak sme našli e-mail v kolekcii Clients, tak pomocou modulu bCrypt testujeme heslo s hešom, ktorý sa nachádza v databáze. Ak sa heš zhoduje s heslom, tak môžeme vrátiť úspešne vykonanie modulu passport-local.

```
done(null,{email: email, firstName: meno,
lastName: priezvisko, role: role, id: id});
```

Tieto dáta dostaneme späť na klienta a pomocou nich môžeme danému užívateľovi udeliť právo, ktoré má v atribúte role.

Pri nesprávnom hesle, alebo keď e-mail ešte nie je verifikovaný, tak jednoducho vrátime neúspešne vykonanie modulu passport-local.

```
done(null,false);
```

4.2.7 Dizajn systému

Pre rezervačný systém sa použil Angular Material Design. Ide o UI komponenty, ktoré sa používajú vo frameworku Angular.js. Túto knižnicu si môžeme stiahnúť prostredníctvom balíčkoveho manažéra Bower. Príkladom môže poslúžiť design správy pre užívateľa.

```
<md-dialog>
      <md-content>{{message}}</md-content>
      <div class="md-actions">
              <md-button ng-click="closeDialog()">
                  OK
              </md-button>
      </div>
</md-dialog>
```

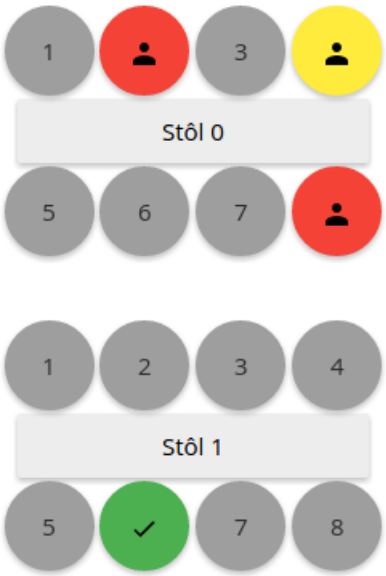
Pre detailnejší náhľad do rôznych iných prvkov nám poslúži dokumentácia na stránke https://material.angularjs.org.

4.2.8 Dynamické generovanie dát

Na zobrazenie stolov a sedadiel používame Angular Material Design. Na strane klienta vytvoríme funkciu na inicializáciu dát. Potrebujeme natiahnúť z databázy všetky potrebné dáta, ktoré sú nutné k zobrazeniu.

```
var Tables = $resource('/api/tables');
var tables = Tables.query({}, function()
  {
     $scope.tables = tables;
  });
```

Na obr. 3 môžeme vidieť zobrazenie stolov a sedadiel.



Obrázok 3: Ukážka stolov a sedadiel

Každú kolekciu si uložíme do premennej $scope. Tento $scope budeme využívať v HTML súbore na napojenie direktív. V HTML súbore začíname prechádzať od kolekcie Rooms.

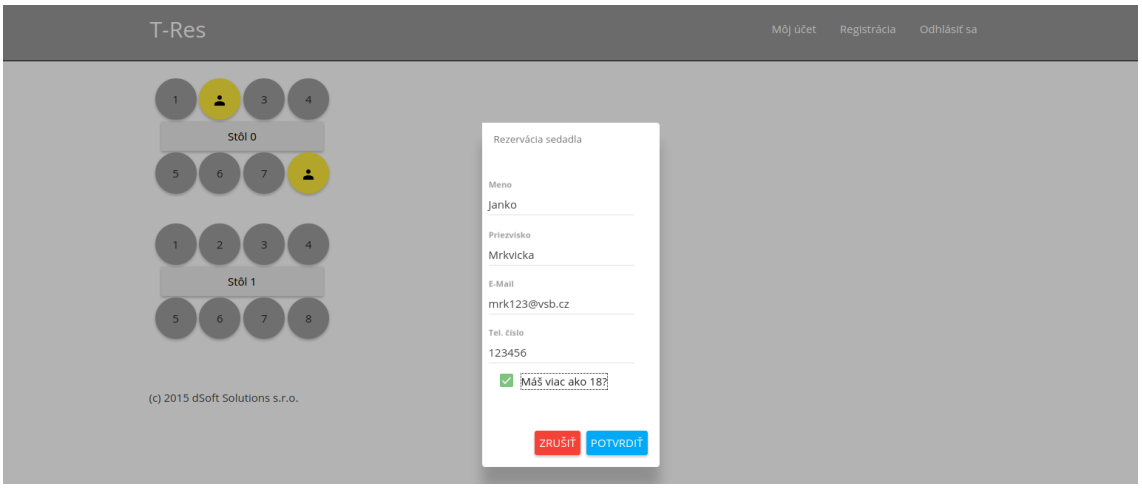
```
<div layout="row" layout-margin layout-fill layout-padding ng
repeat="room in rooms" layout-align="{{room.positioning}}">
```

Následne prechádzame kolekciu Parts, ktorá je referencovaná na kolekciu Rooms. Kolekcia Parts je referencovaná na kolekciu Tables. Kolekcia Tables nám bude zobrazovať stoly. Kolekcia Tables je referencovaná na kolekciu Seats. Kolekcia Seats bude zobrazovať sedadlá k stolom. Pre správne zobrazenie sedadiel je zobrazovanie sedadiel rozdelené na polovicu stola. Zabezpečuje to táto podmienka.

```
<span ng-show="all_table.seats.length" ng-repeat="seat in
all_table.seats"  ng-if="(seat.order-1) <
(all_table.seats.length/2)">
```

Následne je potrebné filtrovať stav sedadla. Môže ísť o **voľný**, **rezervovaný**, **obsadený** alebo **zaplatený** stav. Na základe tejto filtrácie sa nám farebne odlíšia sedadlá a klient bude presne vedieť, ktoré sedadlo si môže rezervovať, a ktoré už nie. Pri prechádzaní kolekciami treba taktiež ošetriť, či klient je prihlásený alebo nie.

Ak nie je prihlásený, tak sa mu sedadlá môžu ukázať len ako voľné alebo obsadené ale ako náhle je prihlásený, tak môže vidieť aj svoje rezervované alebo zaplatené sedadlo. Každé zobrazené sedadlo obsahuje udalosť na kliknutie. Po kliknutí na sedadlo sa zobrazí pomocou Angular Material Design formulár na rezervovanie. Ak je sedadlo v stave obsadené (červená farba), tak sa formulár nezobrazí. Na obr. 4 môžeme vidieť rezervačný formulár.



Obrázok 4: Rezervačný formulár

Pre zobrazenie rezervačného formulára sa používa funkcia show, ktorá preberá controller a HTML súbor.

```
$mdDialog.show
({
   controller: DialogController,
   templateUrl: 'assets/tpl/dialog.tpl.html'
});
```

V controllery riešíme funkciu na potvrdenie rezervácie a funkciu na zrušenie rezervácie. Controller ešte obsahuje cykly, ktoré majú zobrazovať už rezervované dáta. Po vyplnení údajov a potvrdení formulára sa spracuje direktíva ng-submit, ktorá najskôr otestuje či je užívateľ prihlásený alebo nie. Ak je užívateľ prihlásený, tak sa úspešne uložia údaje a zmení sa stav sedadla na rezervovaný (žltá farba). Ak by užívateľ nebol prihlásený, tak ho systém upozorní hláškou a presmeruje na prihlasovací formulár. Užívateľ sa nemusí báť o stratenie vyplnených údajov. Tieto údaje sa uložia do lokálneho úložiska localStorage.

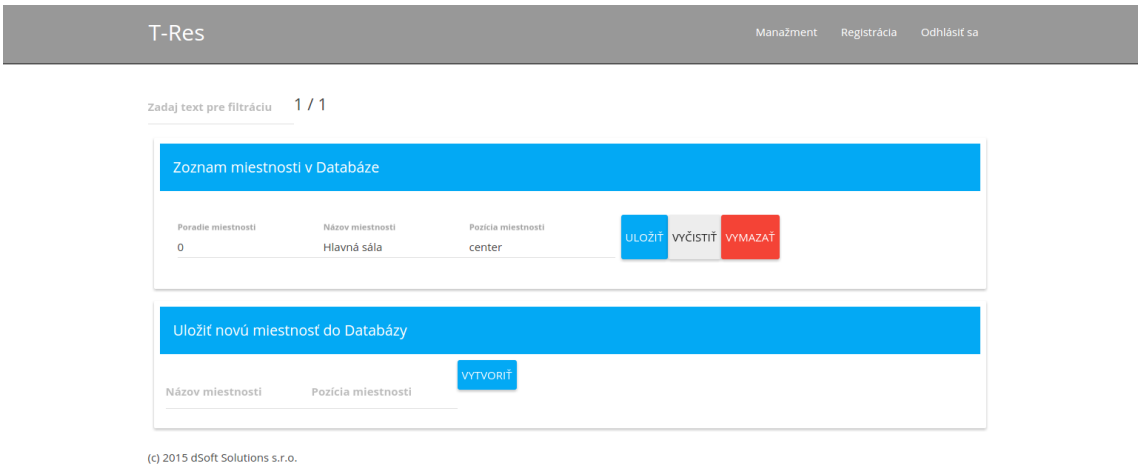
```
localStorage.setItem('firstName',$scope.firstName);
```

Pri presmerovaní na prihlasovací formulár bude k URL pridaný parameter, ktorý vyjadruje, že išlo o objednávku. Týmto zabezpečíme, že po prihlásení bude užívateľ presmerovaný rovno k rezervačnému formulári s údajmi, ktoré vyplnil.

```
$window.location.assign('/login?from='+$location.url(
+"&order=true");
```

4.2.9 Správa systému

Každý administrátor alebo manažér by mal mať možnosť vo svojom účte pridávať, upravovať alebo vymazávať údaje z databázy. Vytvorili sme menu rozhranie, kde si administrátor alebo manažér jednoducho klikne a môže manipulovať s danou kolekciou. Na obr. 5 môžeme vidieť ukážku správy miestnosti.



Obrázok 5: Ukážka správy miestnosti

Pre každú kolekciu si vytvoríme nový HTML a JavaScript súbor. Napríklad pre kolekciu Rooms si vytvoríme rooms.js, ktorý bude obsahovať controller, v ktorom budú funkcie na pridávanie, upravovanie alebo vymazávanie miestnosti. Vytvoríme si aj rooms.html súbor, ktorý bude zobrazovať dizajn všetkých miestností. Dáta sú získané z premennej $scope.

```
var Rooms = $resource('/api/rooms');
$scope.rooms = Rooms.query({});
```

Pre každú operáciu, či už pridávanie, vymazávanie alebo upravovanie sa na strane klienta použije služba $resource. Ďalej sa na strane servera použijú potrebné metódy na vykonávanie operácií.

4.2.10 Pridelenie práv

Administrátor alebo manažér majú možnosť posielať pozvánky na pridelenie roly. Avšak manažér môže prideliť rolu len manažérovi alebo klientovi. Administrátor môže prideliť rolu manažérovi, klientovi ale aj administrátorovi. Pozvánka je realizovaná najskôr HTML formulárom. Po vyplnení údajov môže administrátor alebo manažér pozvánku danému užívateľovi zaslať. Dáta sa uložia do objektu JSON a pošlu sa prostredníctvom služby $resource na serverovú stranu. Na serverovej strane v Express.js aplikácií prejdeme celú kolekciu Clients a hľadáme, či náhodou už daný užívateľ nemá rolu, ktorú mu chceme poslať. Administrátor alebo manažér, ktorý poslal pozvánku môže dostať zo serverovej strany štyri hlášky:

- Pozvánka bola úspešne odoslaná.
- Užívateľ už má danú rolu.
- Užívateľ už má všetky možné roly.
- Užívateľovi už bola odoslaná pozvánka na danú rolu.

Týmto spôsobom bude osoba, ktorá zasiela pozvánku vždy vedieť, aký je stav pozvánky a či už náhodou osoba nemá danú rolu. Ak nájdeme daného užívateľa, ktorý ešte nemá danú rolu, ktorú mu chceme poslať, tak kód pokračuje ďalej. Vytvorí sa nový záznam v kolekcii Confirmations. Kolekcia slúži na uchovávanie informácií o odoslanej pozvánke. Táto kolekcia má následujúce atribúty:

- addressedTo - komu bola pozvánka odoslaná,
- createdBy - kto poslal pozvánku,
- state - aký je stav pozvánky,
- role - aká rola, bude pridelená užívateľovi.

Po vložení záznamu do kolekcie Confirmations sa spracuje štruktúra pozvánkového e-mailu. V e-maily budú zobrazené dva odkazy na prijatie alebo zamietnutie roly.

```
var linkAccept = 'http://' + request.get('host') + '/auth/
invitation?id=' + addressedId+"&role="+role+"&answer=accept";

var linkReject = 'http://' + request.get('host') + '/auth/
invitation?id=' + addressedId+"&role="+role+"&answer=reject";
```

Týmto spôsobom má užívateľ možnosť jednoducho sa rozhodnúť či chce alebo nechce prijať danú rolu. Taktiež má administrátor a manažér možnosť vo svojom účte vidieť všetky odoslané pozvánky a ich stav. Ak nejakým spôsobom užívateľ nedostal pozvánkový e-mail, tak ho môže odosielateľ znova odoslať. Pokiaľ nebola ešte pozvánka prijatá, tak ju môžeme ešte zrušiť. Klient, administrátor alebo manažér má tiež možnosť vo svojom účte vidieť všetky pozvánky, ktoré mu boli poslané. Môže ich prijať alebo zamietnúť.

5 Znalosti a dovednosti získané počas štúdia a uplatnené v priebehu odbornej praxi

Na bakalárskej praxi som najviac využil znalosti z predmetu **Vývoj internetových aplikácií**. V rámci tohto predmetu som mal možnosť naučiť sa technológie ako **JavaScript**, **HTML5**, **CSS3**, ktoré sú potrebné pre vývoj webových aplikácií. Bez znalosti týchto základných technológií je veľmi ťažké začať s vývojom akejkoľvek internetovej aplikácie. Taktiež predmety **Úvod do databázových systémov** a **Databázové a informačné systémy** mi pomohli pri realizácií rezervačného systému. V rámci týchto predmetov som sa naučil, ako pracovať s databázovými technológiami a ako tieto technológie využiť na tvorbu informačných systémov.

Pri úlohe s inštaláciou a konfiguráciou systému Nagios som využil znalosti z predmetu **Počítačové siete**. Týmto predmetom som sa naučil princípy základných počítačových sietí a porozumel som najdôležitejším protokolom používaným v internetu.

Znalosti, ktoré som získal počas štúdia neboli však úplne dostačujúce na to, aby som zvládol odbornú prax vo firme. Musel som sa veľa vecí naučiť sám, aby som bol schopný zvládnuť dané úlohy.

6 Záver

Všetky úlohy, ktoré mi boli pridelené som splnil. Čo sa týka rezervačného systému, tak ešte nie je doimplementovaná časť uhradenia faktúry, pretože firma ešte nemá premyslené, ako to realizovať.

Absolvovanie odbornej praxi v spoločnosti dSoft Solutions, s. r. o. môžem hodnotiť veľmi pozitívne. Vďaka pôsobeniu vo firme som sa naučil riešiť veci efektívnejšie. Svoje znalosti zo školy som obohatil o reálne praktické znalosti z firmy. Naučil som sa taktiež nové technológie, ktoré sme sa ešte v škole neučili. Myslím si, že voľba absolvovania odbornej praxi namiesto vybratia témy je výhodnejšia, pretože pocítite, aké to je pracovať na reálnych projektoch a nadobudnete praktické skúsenosti. Zároveň máte možnosť spojiť doteraz získané teoretické znalosti a dovednosti priamo s praxou.

7 Referencie

[1] AngularJS = MVC framework pre JavaScript [online]. [cit. 2015-03-17]. Dostupné z: www.spireng.sk/content/angularjs-mvc-framework-javascript.

[2] Co je to JavaScript [online]. [cit. 2015-03-17]. Dostupné z: www.adaptic.cz/znalosti/slovnicek/javascript.

[3] dSoft Solutions s. r. o. [online]. [cit. 2015-03-17]. Dostupné z: www.dsoft.eu.

[4] Express - Node.js web application framework [online]. [cit. 2015-03-17]. Dostupné z: www.expressjs.com.

[5] Git [online]. [cit. 2015-03-17]. Dostupné z: www.git-scm.com.

[6] MongoDB [online]. [cit. 2015-03-17]. Dostupné z: www.mongodb.org.

[7] Nagios [online]. [cit. 2015-03-17]. Dostupné z: www.nagios.org.

[8] Node.js [online]. [cit. 2015-03-17]. Dostupné z: www.nodejs.org.